Using MPIProf for Performance Analysis

NAS Webinar

July 20, 2016

NASA Advanced Supercomputing Division

Outline

- What is MPIProf and Why
- Basic usage of the `mpiprof` tool
- Profile results explained (Overflow as an example)
- Used-defined profiling via the `mprof` API
- Accuracy and overhead study

Performance Analysis

- Understanding performance characteristics of applications
  - Important for
    - Optimizing application performance to reduce compute time
    - Improving computing resource utilization
- Performance analysis tools
  - Often required due to
    - Sophistication in modern high performance computing systems
      - Hierarchical architecture with multicore CPUs and accelerators
      - Sophisticated memory system and network
    - Complicated application structure
  - Commercial tools
    - Intel Vtune, Allinea MAP, ITAC, SGI MPInside, IOT, op_scope, etc.
  - Open-source, research tools
    - TAU, OpenSpeedshop, PerfSuite, etc.

What is MPIProf?

- A profile-based application performance analysis tool
  - Gathers statistics in a counting mode
  - Reports aggregated and per-rank profiling information
  - Supports user-defined profiling
  - Works with many MPI implementations
    - including SGI MPT, Intel MPI, MPICH, MVAPICH, and OpenMP
- Reporting profiling information about
  - Point-to-point and collective MPI functions called by an application
    - time spent, number of calls, message size
  - MPI I/O and POSIX I/O statistics
  - Memory used by processes on each node
  - Call-path based information

Question? Use the Webex chat facility to ask the Host

Why MPIProf?

- Simple interface
  - A command-line tool without the need of modifying or recompiling applications
  - Auto-detection of MPI environment from different implementations
  - Text output with tabulated results for easy post processing
  - User-defined profiling only if needed

- Lightweight approach
  - Counting based, small amount of data
  - Low overhead (in both data collection and memory usage)

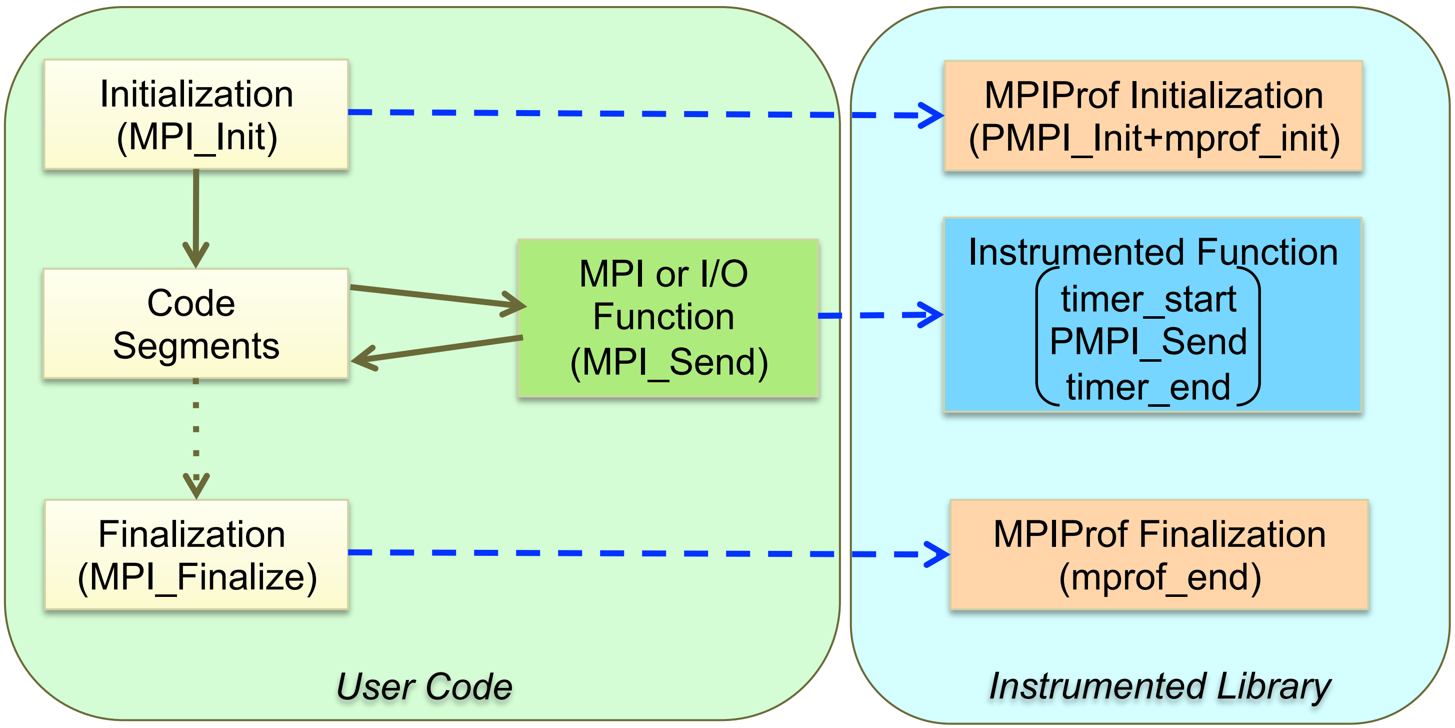
Question? Use the Webex chat facility to ask the Host

MPIProf Basics

Question? Use the Webex chat facility to ask the Host

Instrumentation Infrastructure

- Instrumenting MPI or I/O functions
  - By the PMPI interface and `dlsym` for dynamic shared library
  - Accessing instrumented functions via `LD_PRELOAD` or linking with the instrumented library
  - Call-path information provided by the `libunwind` interface



What're Monitored?

- MPI Functions (MPI 3)
  - Point-to-point calls (blocking and nonblocking)
    - MPI_Send, MPI_Recv, MPI_Isend, MPI_Irecv, MPI_Wait, etc.
  - Collective calls (blocking)
    - MPI_Bcast, MPI_Gather, MPI_Reduce, MPI_Allgather, MPI_Allreduce, etc.
  - Collective calls (nonblocking)
    - MPI_Ibarrier, MPI_Ibcast, MPI_Igather, MPI_Ireduce, MPI_Wait, etc.
  - One-sided communication calls
    - MPI_Put, MPI_Get, MPI_Accumulate, MPI_Win_complete, etc.
  - MPI I/O calls
    - MPI_File_open, MPI_File_read, MPI_File_write, etc.

What're Monitored?

- POSIX I/O calls
  - open
    - open | fopen | creat | open64 | creat64
  - close
    - close | fclose
  - read
    - read | fread | pread | pread64
  - write
    - write | fwrite | pwrite | pwrite64
  - fsync
    - sync | fsync | fdatasync

Reported Profiling Results

- Summary section
  - Timing for communication, blocking, I/O, and computation
  - Message size and rates
  - I/O size and rates
  - Memory usage of processes on each node
  - Per-function summary
- Break-down results in each rank
  - Timing, number of calls, message size, I/O size
- Map of messages communicated among ranks
  - Rank, timing, message size
- Call-path information
  - Timing along call-path for the instrumented functions

Two Types of Usage

- The **`mpiprof`** profiling tool
  - Whole program analysis
  - No change or recompilation of application

```
mpiexec -np <n> mpiprof [-options] a.out [args]
```

- The **`mprof`** API routines
  - Selective profiling for selected code segments
  - Requires modification of application (instrumentation)
    - Link with the **`mprof`** library, run as normal
- Control of the amount of profiling information
  - Via **`mpiprof`** options
  - Via environment variables
  - See the user guide for details

Accessing MPIProf

- Load the proper modules

```
module load comp-intel/2016.2.181
module load mpi-sgi/mpt.2.12r26
module load /u/scicon/tools/modulefiles/mpiprof-module
```

  - The latest **`mpiprof`** version is 1.8.2

- Run your code

```
mpiexec –np 64 mpiprof a.out
```

  - Results will be written to "`a.out_64_mpiprof_stats.out`" at the end of a run

```
mpiprof a.out
```

  - For serial (non-MPI) codes

- To get a quick help on **`mpiprof`** options, use

```
mpiprof -help
```

http://www.nas.nasa.gov/hecc/support/kb/using-mpiprof-for-performance-analysis_525.html
pfe:/u/scicon/tools/opt/mpiprof/doc/mpiprof_userguide.pdf

The `mpiprof` Tool and Options

Question? Use the Webex chat facility to ask the Host

The `mpiprof` Profiling Tool

- Functionality
  - Whole program analysis
  - No change or recompilation of an application
  - The [-g] compilation flag recommended if collecting call-path information

- Basic usage
  - For MPI codes

    ```
    mpiexec –np <n> mpiprof [-options] a.out [args]
    ```

  - For non-MPI codes

    ```
    mpiprof [-options] a.out [args]
    ```

- Control of profiling information
  - Via command options or environment variables

Question? Use the Webex chat facility to ask the Host

`mpiprof` Options

Option	Description
`-lib` *<mproflib>*	selects a runtime profiling library *<mproflib>*
`-o` *<outfile>*	writes profiling results to *<outfile>*
`-[c,p]blk`	estimates blocking time for collective and/or point-to-point communication calls
`-msgm`	collects rank-based message size and count maps
`-byte`	prints message size in bytes
`-pflag` *<value>*	sets `MPROF_PFLAG` to *<value>*
`-mfunc` *<func:n>*	specifies a function to be monitored
`-csig[=`*<signo>*`]`	writes output stats when a signal is caught
`-mem`	reports memory usage only
`-ios`	reports I/O statistics and memory usage only
`-cpath[=`*<depth>*`]`	collects call-path information
`-expr=`*<exps>*	performs cpu+comm scaling experiments (experimental)
`-v`	sets verbose flag

Question? Use the Webex chat facility to ask the Host

Env Variable `MPROF_PFLAG`

`MPROF_PFLAG=`*<value>*

<value>	**Description**
`disable`	disables profiling environment
`off | false`	switches off profiling
`on | true`	switches on profiling
`cblk`	estimates collective blocking time
`pblk`	estimates point-to-point blocking time
`blk`	is equivalent to "`cblk+pblk`"
`msgm | msgmx`	collects message size and count maps
`byte`	prints message size in bytes
`mem`	reports memory usage only
`ios`	reports I/O statistics and memory usage only
`cpath | cpathx`	collects call-path information

Note: **`mpiprof`** *options override the value of* ***`MPROF_PFLAG`***

Use of `mpiprof` Options

- Profiling in default setting (without other options)
  - Included
    - MPI functions, POSIX I/O functions
    - Memory usage
  - Not included
    - Blocking time measurement for MPI calls
    - Rank-based message maps
    - Call-path information
- A few useful options

`-cblk`	to enable blocking time measurement for collective calls
`-msgm`	to enable report of rank-based message maps
`-cpath`	to enable call-path information collection
`-byte`	to report message size and I/O size in bytes
`-mem`	to report memory usage only without profiling
`-ios`	to report I/O stats only (no MPI functions)

Question? Use the Webex chat facility to ask the Host

Profile Results Explained

Question? Use the Webex chat facility to ask the Host

The Overflow Test Case

- The NTR benchmark test case
  - DLRF6, 36 million grid points
  - 128 MPI processes on 8 Intel Sandy Bridge nodes

- Two run setups
  - Using the default setting

```
mpiexec -np 128 mpiprof ./overflowmpi
```

  - Measuring the blocking time from MPI collectives

```
mpiexec -np 128 mpiprof -cblk ./overflowmpi
```


Question? Use the Webex chat facility to ask the Host

Sample Outputs

```
MPIPROF v1.8.2, built 06/30/16, collected 06/30/16 09:35:17

==> List of environment variables
   MPROF_LIB    = sgimpt
   MPROF_EXEC   = ./overflowmpi

==> Summary of this run
   Number of nodes             = 8
   Number of MPI ranks         = 128
   Number of inst'd functions  = 17

   Total wall clock time       = 1027.71 secs
   Average computation time    = 858.698 secs (83.55%)
   MPIProf overhead time       = 0.09571 secs ( 0.01%)

   Average communication time  = 168.665 secs (16.41%)
      collective               = 96.6505 secs ( 9.40% or 57.30%Comm)
      point-to-point           = 72.0143 secs ( 7.01% or 42.70%Comm)
   Total message bytes sent    = 1.5020T
      collective               = 40.252G
      point-to-point           = 1.4617T
   Total message bytes received = 1.5020T
      collective               = 40.252G
      point-to-point           = 1.4617T
   Gross communication rate    = 10.0448 Gbytes/sec
   Communication rate per rank = 78.4749 Mbytes/sec

   Average I/O time (%, L, H)  = 0.25426 secs ( 0.02%, 0.00000, 32.5237)
      write time               = 0.21419 secs ( 0.02%, 0.00000, 27.3985)
      read time                = 0.04007 secs ( 0.00%, 0.00000, 5.12879)
   . . . . . .
```

Section header in the report

Summary of Profile Results

- Statistics about a run
  - Number of nodes, ranks, and instrumented functions
  - Overall timing and rate information
- Meanings of a few key entries

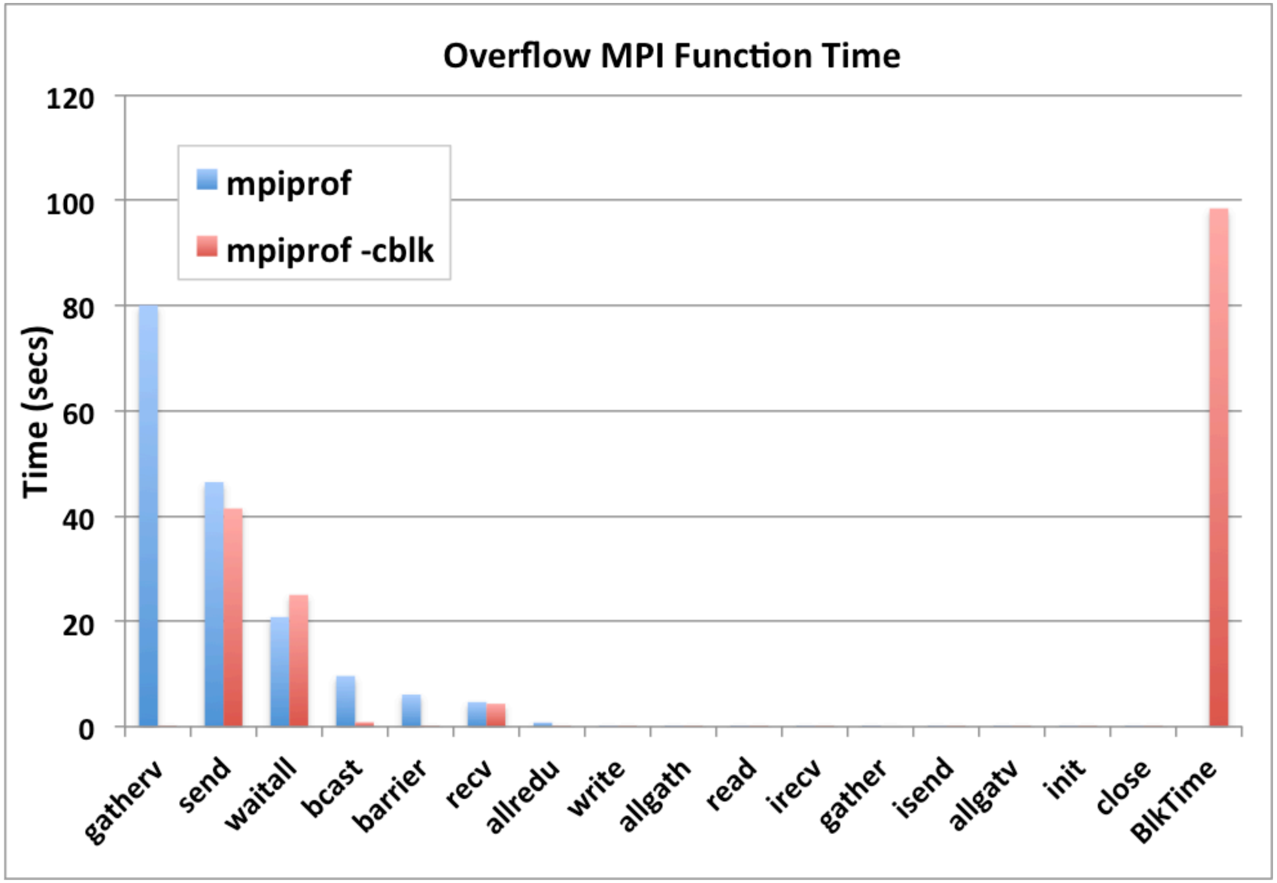
Entry	Symbol	Description
`Total wall clock time`	T(wallclock)	Time spent from `MPI_Init` (inclusive) up to `MPI_Finalize`, or from `mprof_init` to `mprof_end`
`Average computation time`	T(comp)	= T(wallclock) − T(comm) − T(i/o) − T(overhead)
`MPIProf overhead time`	T(overhead)	Average time used by MPIProf for gathering data, including `mprof_init` but excluding `mprof_end`
`Average communication time`	T(comm)	Average time spent in MPI calls, excluding MPI-IO
`Average I/O time`	T(i/o)	Average time spent in MPI-IO and Posix I/O
`Effective I/O time`	T(eff_i/o)	Time estimated from I/O rates for each rank
`Communication rate`	r(comm)	= Message size / t(comm) for each rank
`I/O rate`	r(i/o)	= Data size / t(i/o) for each rank

Reported Information

- In summary sections
  - Average time across all ranks for communication, I/O, computation
  - Percentage of time relative to the total wall clock time
  - Communication and I/O rates
    - Calculated for each rank
    - Aggregated for all ranks
  - Memory usage
- Per-function summary
  - List of instrumented functions
  - Break-down timing, counts, message/data size
- Messsage/data size histograms
- Per-rank profiling data
  - Break-down timing, counts, message/data size

Question? Use the Webex chat facility to ask the Host

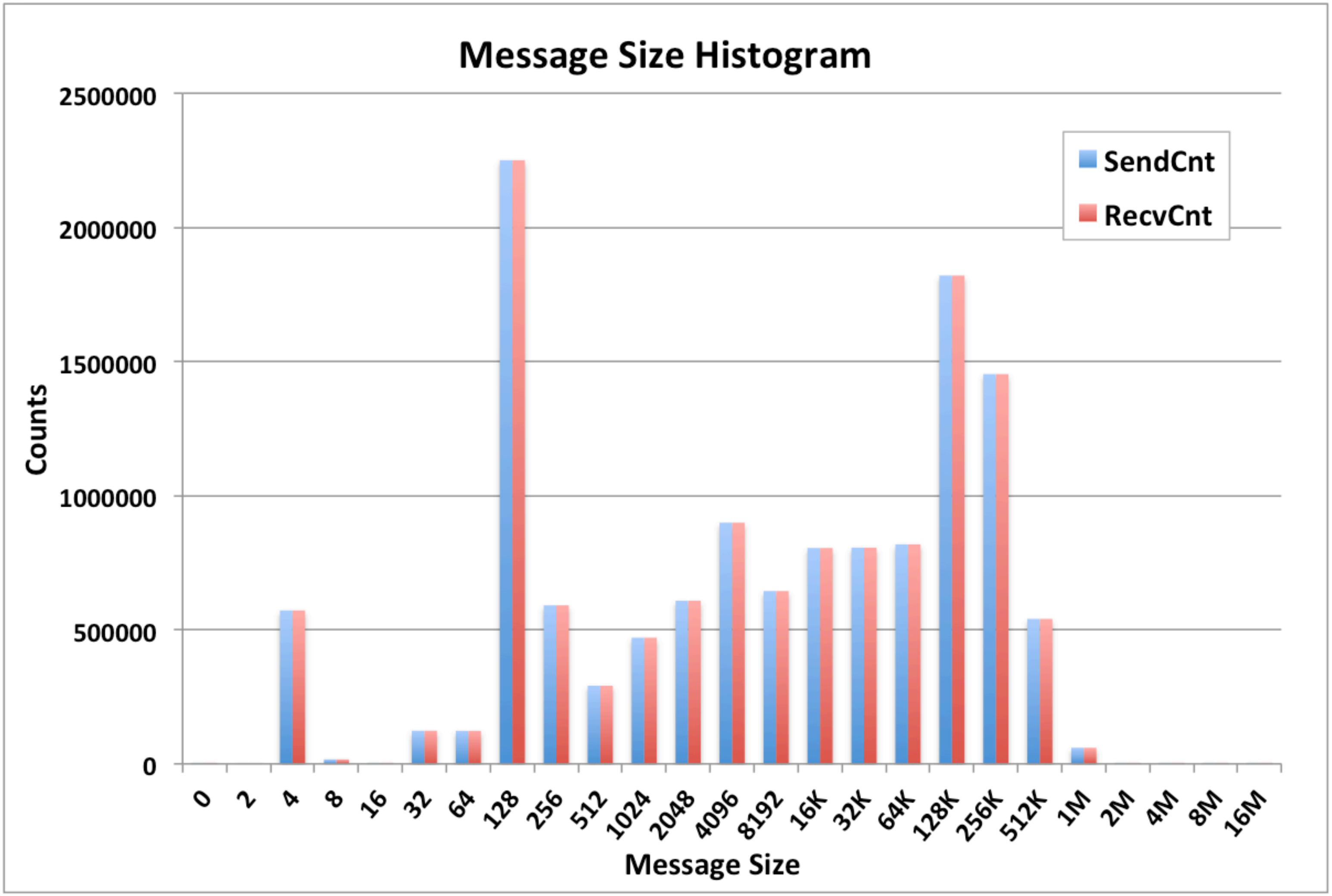
Per-Function Timing



- Sorted by timing from the run with default setting

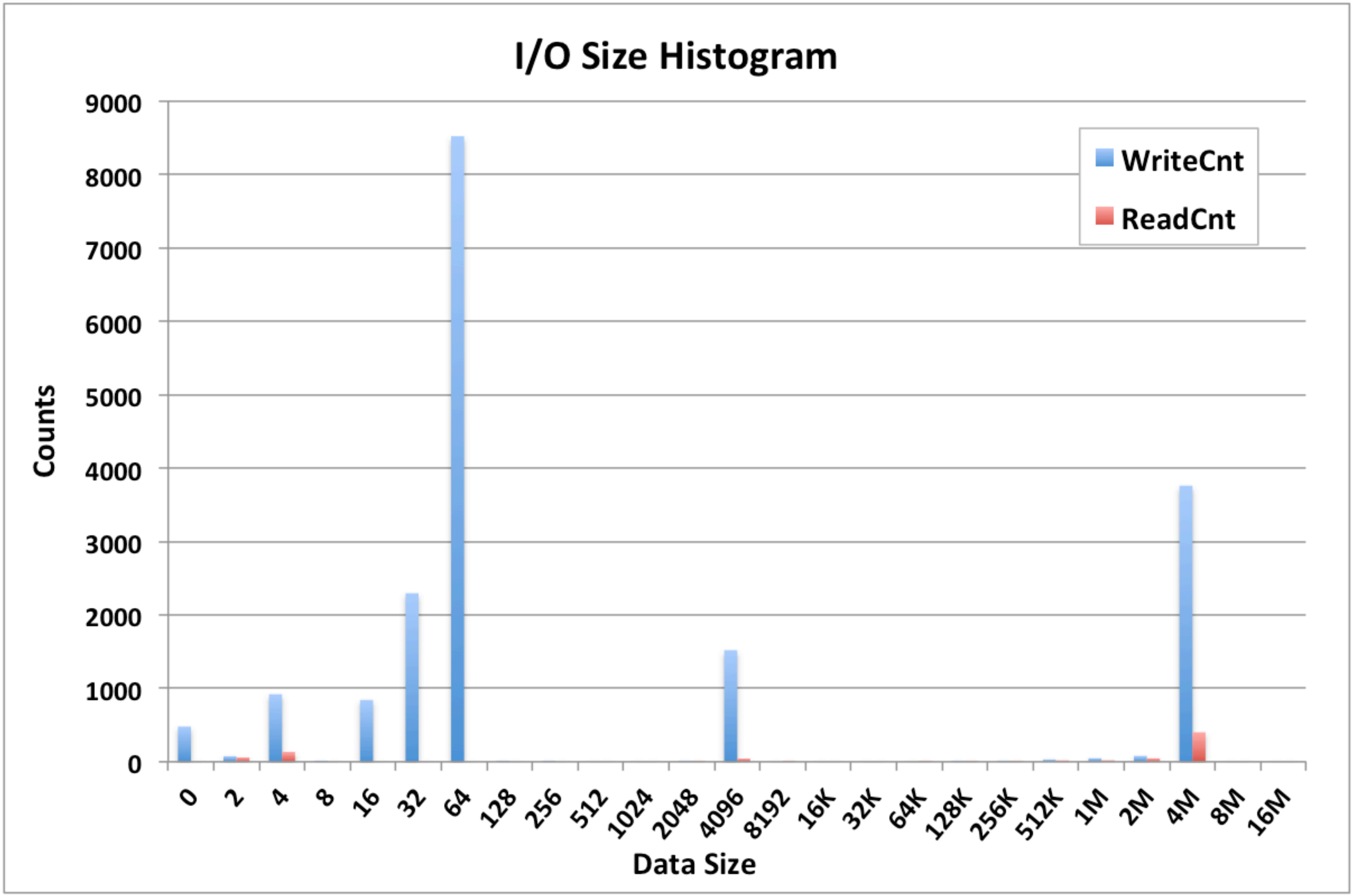
Question? Use the Webex chat facility to ask the Host

Message Size Distribution



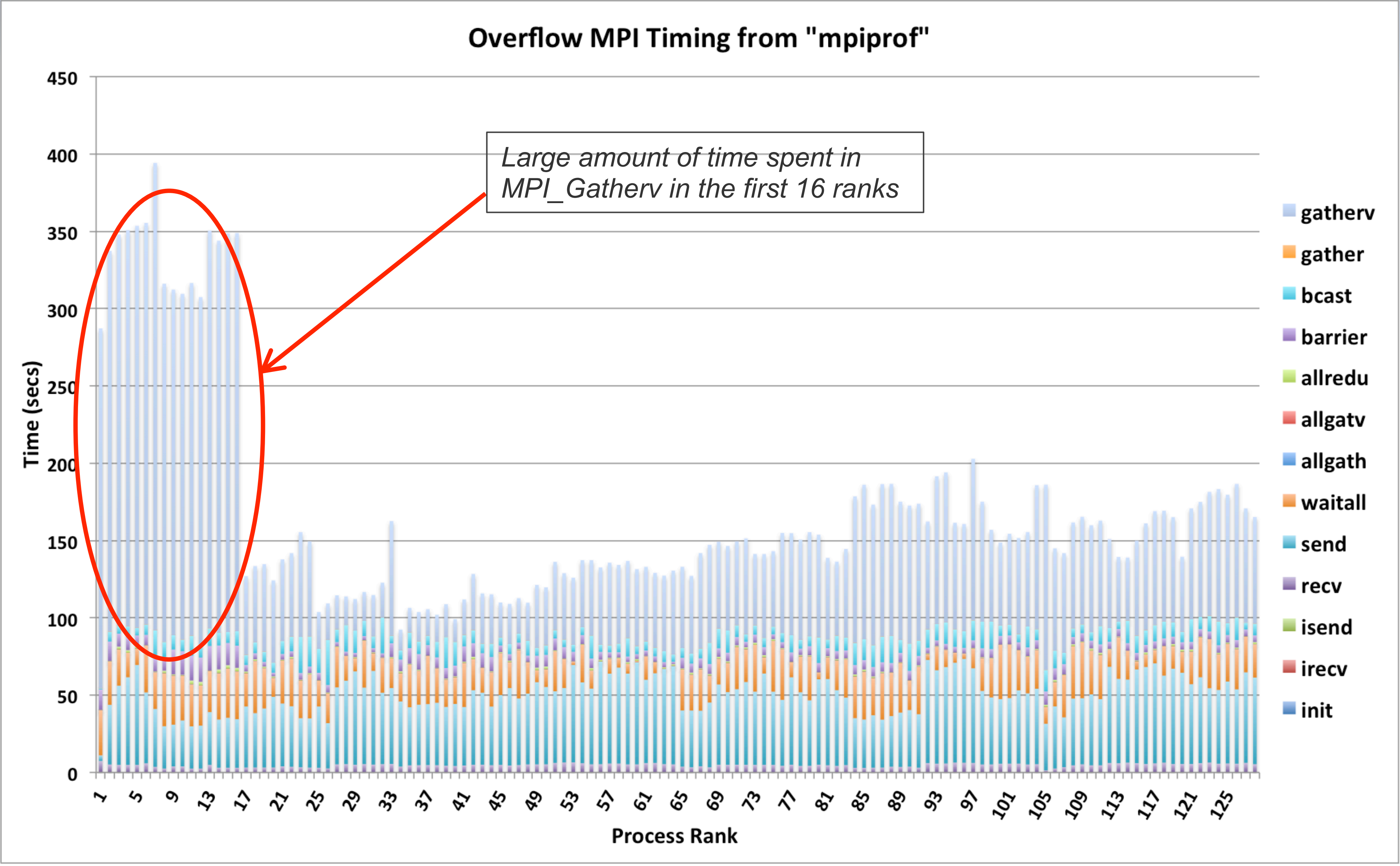
Question? Use the Webex chat facility to ask the Host

I/O Data Size Distribution



Question? Use the Webex chat facility to ask the Host

Function Profiling on Each Rank



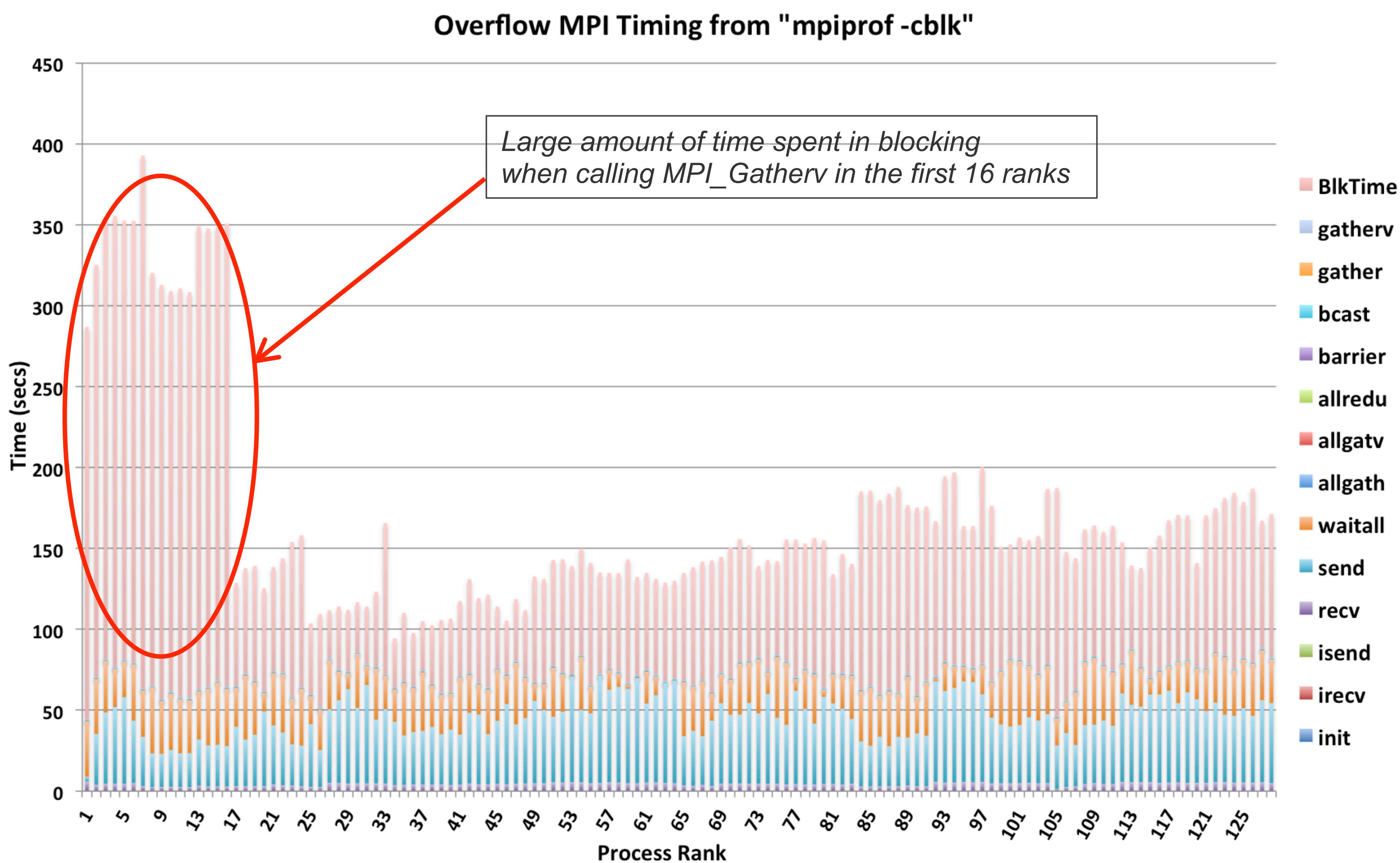
Question? Use the Webex chat facility to ask the Host

Blocking Time in MPI Calls

- Two parts of time in MPI calls
  - Time waiting for the post of a message from a remote rank
  - Actual time spent in transmitting the message
- A large waiting (or *blocking*) time
  - Usually an indication of load imbalance
- Measurement of blocking time
  - No direct measurement without knowing the MPI implementation details
  - Estimation of blocking time by MPIProf (via the `-cblk` option)
    - Inserting a barrier in front of each collective call
    - Measuring time spent in the barrier
    - Reporting the effective communication time (excluding the blocking time)

Question? Use the Webex chat facility to ask the Host

Function Profiling with "Blocking"



Question? Use the Webex chat facility to ask the Host

I/O Stats Summary

- Averaged I/O stats
  - Calculated across all ranks
- Effective I/O stats
  - Estimated from rates

    T_effective = total_IO_data / aggregated_IO_rate
  - Useful to show when I/O is unbalanced
- Example

L: minimum on a rank, H: maximum on a rank

```
Average I/O time (%, L, H)    = 0.25426 secs ( 0.02%, 0.00000, 32.5237)
   write time                 = 0.21419 secs ( 0.02%, 0.00000, 27.3985)
   read time                  = 0.04007 secs ( 0.00%, 0.00000, 5.12879)
Effective I/O time (%, iF)    = 32.3911 secs ( 3.15%, 126.39)
   effective write time       = 27.2623 secs ( 2.65%, 126.28)
   effective read time        = 5.12879 secs ( 0.50%, 127.00)
```

imbalance factor (iF) = (T_effective – T_average) / T_average

More on I/O Stats

- I/O stats from option [-ios]
  - Do not include MPI I/O calls, but rather from low level I/O calls
  - May show different behavior than without the option
  - Reflect more about details of an MPI implementation
- Example: the FLASH IO benchmark with collective MPI I/O
  - 120-rank run across 5 nodes with 24 ranks/node
  - SGI MPT with Lustre filesystem support
  - I/O stats from [-ios]
    - With a stripe count of 1, only rank 0 does the writes
    - With a stripe count of 12, 4 ranks (0,24,48,72) do the writes
  - Related to the optimization made by MPT for I/O collective buffering

```
if (#nodes >= #stripes)
   #io_ranks = #stripes
else
   #io_ranks = largest number < #nodes that evenly divides #stripes
```

User-Defined Profiling Interface (`mprof` Routines)

The `mprof` API Routines

- For profiling selected code segments
  - Instrumentation manually
  - Code recompilation required

- Four **`mprof`** API routines

  `mprof_init(pflag)` - initializes the profiling environment, all ranks
  `mprof_start()` - switches on data collection
  `mprof_stop()` - switches off data collection
  `mprof_end()` - finalizes and writes stats to output, all ranks

- Header include files
  - For C: `include “mprof_lib.h”`
  - For Fortran: `include “mprof_flib.h”` or `use mprof_flib`

Question? Use the Webex chat facility to ask the Host

API Routine Calling Sequence

Fortran Example

```
include "mprof_flib.h”  (or use mprof_flib)

! initialize and turn on profiling
call mprof_init(MPF_ON)
... 1st profiled code segment
call mprof_stop()  ! stop profiling

... Code segments without profiling

call mprof_start() ! restart profiling
... 2nd profiled code segment
call mprof_end()   ! Finish and write results
```

`MPF_ON` *implies* `mprof_start`

Repeat as needed

Implied `mprof_stop`

Question? Use the Webex chat facility to ask the Host

Compilation and Execution

- Prerequisite: Load proper modules

```
module load mpiprof-module
```

- Flags for compilation
  - At compiling time

    `-I${MPROF_DIR}/include` (or `-I${MPROF_INC}`)
  - At linking time

    `-L${MPROF_DIR}/lib -lmpiprof_<mproflib> [-lmprof_flib]`

    `<mproflib>` is one of the supported libraries (`sgimpt`, `intelmpi`, …)
- At runtime
  - Run the instrumented code with or without the `mpiprof` tool
  - For the latter case, use environment variables to control profiling
    - Such as setting `MPROF_LIB=sgimpt`

MPIProf Accuracy and Overhead

Question? Use the Webex chat facility to ask the Host

Accuracy and Overhead

- Runtime overhead
  - In initial setup
  - During data collection for each function
  - In final writing of output data
- Memory overhead
  - From MPIProf internal buffers
- Accuracy and overhead study
  - Use NPB3.3.1-MPI, compare with benchmark timers
  - Three experiments
    - Without `mpiprof`
    - With `mpiprof` in default setting
    - With `mpiprof -cpath` option

Question? Use the Webex chat facility to ask the Host

Accuracy and Overhead Study

- Timing in seconds for seven NPBs
  - Class C problem
  - 64 ranks on Pleiades SandyBridge nodes

	no `mpiprof`		with `mpiprof`			with `mpiprof -cpath`		
Benchmark	Bmk-Time	Bmk-Comm	Bmk-Time	Bmk-Comm	mpiprof-Comm	Bmk-Time	Bmk-Comm	mpiprof-Comm
bt.C.64	18.9623	1.8871	18.8956	1.9270	1.9089	19.0324	1.9993	1.9696
cg.C.64	4.8216	1.5901	4.8298	1.6629	1.6765	4.9218	1.7533	1.6244
ft.C.64	6.3641	2.4831	6.4016	2.5216	2.6989	6.3393	2.4657	2.6104
is.C.64	0.5781	0.3517	0.5966	0.3704	0.4254	0.5920	0.3653	0.4882
lu.C.64	16.2089	2.8413	16.3658	2.9973	2.4488	17.5130	4.2008	2.7898
mg.C.64	1.4375	0.1477	1.4569	0.1656	0.1851	1.4793	0.1908	0.2518
sp.C.64	18.9689	2.6778	18.4860	2.4800	2.4303	18.6505	2.6977	2.6114

- A few observations
  - Difference of benchmark time with/without `mpiprof` is less than 3%
  - The `-cpath` option has slightly larger overhead
  - Measured communication times agree in general with those reported by benchmarks except for LU where "Bmk-Comm" includes time for data packing and unpacking

Memory Overhead

- Memory usage of MPIProf internal buffers
  - Dependent on the number of ranks (N) and the number of instrumented functions (M)
  - An estimate of the buffer memory usage (in bytes)
    - For rank=0: $(656+32*M)*N+5984$
    - For rank>0: $32*(M+N)+5488$
- Examples
  - For a case of N=4096, M=12
    - mem(rank=0) = 4.266 MB
    - mem(rank>0) = 0.137 MB
  - For a case of N=10K, M=15
    - mem(rank=0) = 11.366 MB
    - mem(rank>0) = 0.326 MB

Disclaimer

Limitations

- In data collection
  - No detailed trace information (due to counting mode)
  - For multi-threaded MPI, only the stats from the master thread of each rank are reported
  - When [-ios] is used, MPI I/O information is reported as level-low I/O
  - I/O support still in progress
- Presentation
  - Text-based tool, no GUI support
- Implementation
  - No call-path support for MVAPICH

Question? Use the Webex chat facility to ask the Host

Acknowledgment

- Members of the NAS APP group
  - For constantly testing and sending feedbacks
  - In particular Sherry Chang for constantly requesting new features and patiently reviewing the user guide
- Michael Raymond of SGI
  - For sharing some of the insight of SGI MPT data
- Some of the original idea was motivated by SGI MPInside
- Contact information

  Henry Jin <hjin@nas.nasa.gov>

Question? Use the Webex chat facility to ask the Host